

# Machine Learning and Options Pricing: A Comparison of Black-Scholes and a Deep Neural Network in Pricing and Hedging DAX 30 Index Options

Lauri Stark

Student Number: 484862

Department of Finance  
Aalto University School of Business  
Instructor: Matthijs Lof

December 5, 2017

## **Abstract**

In this paper I study whether a deep feedforward network model performs better than the Black-Scholes model in pricing and delta hedging European-style call options. I apply the methodologies from selected original works on daily prices of call options written on DAX 30 between years 2013 and 2017. My results are mostly consistent with earlier literature and they indicate that the out-of-sample pricing performance of the neural network is superior to Black-Scholes with medium-term and long-term maturities and the out-of-sample delta-hedging performance of the neural network is superior with out-of-the-money options.

## Contents

<b>List of Figures</b>	<b>ii</b>
<b>List of Tables</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 A Brief Overview of Neural Networks</b>	<b>2</b>
2.1 Artificial Neural Networks . . . . .	2
2.2 Multilayer Perceptron . . . . .	3
2.3 The Backpropagation Algorithm . . . . .	5
2.3.1 The Forward Pass . . . . .	5
2.3.2 Parameter Optimization with Stochastic Gradient Descent . . . . .	6
2.4 Neural Networks in Option Pricing . . . . .	7
<b>3 The Data</b>	<b>7</b>
<b>4 Model Specification</b>	<b>9</b>
4.1 Neural Network Structure . . . . .	9
4.2 Performance Measures . . . . .	9
4.2.1 Pricing Performance Measures . . . . .	9
4.2.2 Delta Hedging Performance Measure . . . . .	11
<b>5 Empirical Results and Analysis</b>	<b>12</b>
5.1 Pricing Performance Results . . . . .	12
5.2 Delta Hedging Performance Results . . . . .	15
<b>6 Conclusions</b>	<b>17</b>
<b>7 References</b>	<b>18</b>

## List of Figures

1 The perceptron. . . . .	3
2 An LTU with an input vector of three elements. . . . .	3
3 Multilayer perceptron. . . . .	4
4 Graphs of the data. . . . .	8
5 Estimated volatility and risk-free rate between 1/1/2013 and 19/9/2017 . . . . .	10
6 Pricing error scatter plots. . . . .	14
7 Absolute pricing error surfaces. . . . .	14
8 Black-Scholes delta and neural network delta. . . . .	16
9 $\Delta_{NN}$ from simulated Black-Scholes prices. . . . .	16

List of Tables

1	Pricing performance: whole test set . . . . .	12
2	Delta hedging performance: paired t-tests for the absolute tracking errors for the whole test set. . . . .	15

# 1 Introduction

The theory of options pricing is strongly based on the seminal articles by Black and Scholes (1973) and Merton (1973). In these articles, closed-form solutions for option prices are obtained by using a dynamic hedging strategy and a no-arbitrage requirement. The Black-Scholes formulas are typical examples of traditional parametric, no-arbitrage pricing formulas, and the derivations of these kinds of formulas depend heavily on the assumptions and knowledge of the stochastic process that determines the price dynamics of the underlying asset of the option. If this stochastic process is somehow wrongly specified in the modelling, the following model will yield pricing errors. One solution to this problem is to use data-driven, nonparametric methods such as neural networks.

Several previous studies have compared the performance of Black-Scholes and neural network-based methods in options pricing, most of them with promising results. Hutchinson et. al. (1994) studied the hedging performances of three neural network models with American-style S&P 500 index futures calls and found that all of the neural network models were superior. They also showed that the neural network models can learn the Black-Scholes pricing formula itself up to a high degree accuracy from simulated data. Anders et. al. (1996) compared the pricing accuracy of a neural network with European-style calls written on DAX 30 and also found that the neural network was superior. Amilon (2003) compared both the pricing and hedging performances of a neural network with European-style calls on OMX index with both implicit and historical volatilities and found that the neural network was superior in both cases. Bennell and Sutcliffe (2004) compared the pricing accuracy with FTSE 100 call options to dividend-adjusted Black-Scholes formula and found that the neural network was superior. Different results from these were obtained by Hanke (1999) whose analysis indicated that the performance of Black-Scholes became superior after the parameters for volatility and risk-free rate were optimized. Also, Malliaris and Salchenberger (1993) studied the pricing performance of a neural network model with American-style S&P 100 call options found that the neural network model was superior for out-of-the-money options but for in-the-money options Black-Scholes was superior. Yao et. al. (2000) found that the neural network has better performance when the volatility is high but that the performance of Black-Scholes is generally better for at-the-money options. Generally, the results of the previous studies indicate that neural networks are indeed capable of learning pricing formulas up to high accuracy from real market data.

However, most of the previous studies have been performed with datasets from the 1990s. Moreover, the level of available computational resources has increased significantly during the last 20 years and this makes it possible to build deeper<sup>1</sup>, more computationally-intensive data-driven models such as *deep neural networks*. Therefore, it is of interest to study the performance of such a deep neural network model in option pricing with recent data and with more computational power.

This paper provides a comparison of the pricing and hedging performance of a deep neural network model and the Black-Scholes model and tests whether the results of the previous studies can be reproduced with more recent data and a deeper model. The main hypotheses are based on the previous literature and they state that the deep neural network outperforms the Black-Scholes model 1) in pricing performance and 2) in delta hedging performance. In the analysis, I will apply the methodology of selected original works to a dataset of daily closing prices of DAX 30 call options for the time period 2013–2017. For pricing performance, I will apply the methodologies used by Anders et.

---

<sup>1</sup>The following chapters will clarify what is meant by *deep* in the context of machine learning. Generally, deep models are larger and more complex compared to shallower models.

al. (1996) and Bennell and Sutcliffe (2004). For delta hedging performance, I will follow Hutchinson et. al. (1994).

The structure of this paper is the following. First, I will provide a brief overview of neural networks in order to make it easier for readers without machine learning experience to follow the analysis. Second, I will describe the data and how it was processed. Next, I will specify the architecture of the neural network model that is used in the analysis as well as the performance measures for pricing and delta hedging. After this, I will analyze the empirical results. Finally, I will provide a conclusion of the results and suggestions for further research.

## 2 A Brief Overview of Neural Networks

### 2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are architectures which are designed to mimic the way the human brain works. They were first introduced in the seminal article by McCulloch and Pitts (1943) where the authors presented a model of how real biological neurons are linked and work together in the brains of mammals.

Probably the simplest ANN model is the *perceptron* (Figure 1), presented by Rosenblatt (1958). The building block of the perceptron is a specific type of an artificial neuron, called the *linear threshold unit* (LTU). The LTU (Figure 2) can be thought of as a function which takes a numerical vector input, then calculates a weighted sum of the input and applies some *step function* such as the heaviside function (1) or the sign function (2) to this sum and outputs a binary-like value, depending on the used step function. A perceptron is basically composed of the input vector and one layer of parallel LTUs where each LTU is connected to all of the input vector elements. The output of the perceptron is then a vector whose elements are binary-valued. However, as the perceptron is a really simple model it was quickly noticed that it has several restrictions and weaknesses. These lead to the development of a more advanced model called the *multilayer perceptron* (MLP) which is based on the work of Fukushima (1980).

$$heaviside(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases} \quad (1)$$

$$sgn(z) = \begin{cases} -1, & z \leq 0 \\ 0, & z = 0 \\ 1, & z \geq 0 \end{cases} \quad (2)$$

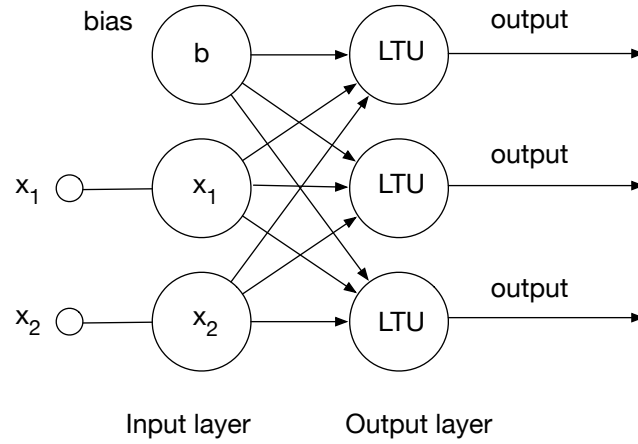


Figure 1: The perceptron.

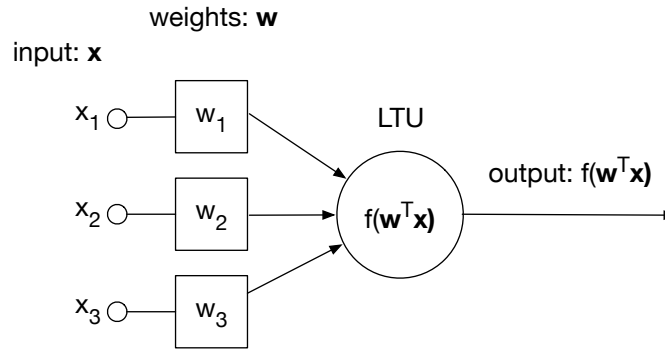


Figure 2: An LTU with an input vector of three elements.

## 2.2 Multilayer Perceptron

As the name suggests, multilayer perceptron (Figure 3), also called as *feedforward network*, consists of multiple layers of neurons. The first layer is called the *input layer*, the last layer is the *output layer* and the layers between the input and output are called the *hidden layers*.<sup>2</sup> Generally, the basic idea in the MLP is really similar to the perceptron. The input layer consists of the input vector and a bias term, and the following layer is the first hidden layer. Each of the neurons of the first hidden layer is connected to all input elements and to the bias term, and the input of each neuron in the first hidden layer is a weighted sum of all the input elements plus the input bias term (the bias is not included in the weighted sum).<sup>3</sup> However, the neurons in the MLP differ from the neurons of the perceptron. The main cause of the restrictions of the perceptron is the linearity of the step functions, and this is the reason why the neurons of MLP apply nonlinear *activation functions* such as logistic sigmoid (3), hyperbolic tangent (4) or different kinds of rectifier linear units (ReLUs) which have increased in popularity recently. The formulation of a basic ReLU is shown in (5). Now, instead of a binary output, each neuron in the first hidden layer outputs a real-valued number depending on the activation function, and this output is then passed to all neurons in the next hidden layer with another bias term. The process continues similarly through all hidden layers, after which the output layer gives the output

<sup>2</sup>A multilayer perceptron is said to be *deep* if it includes two or more hidden layers.

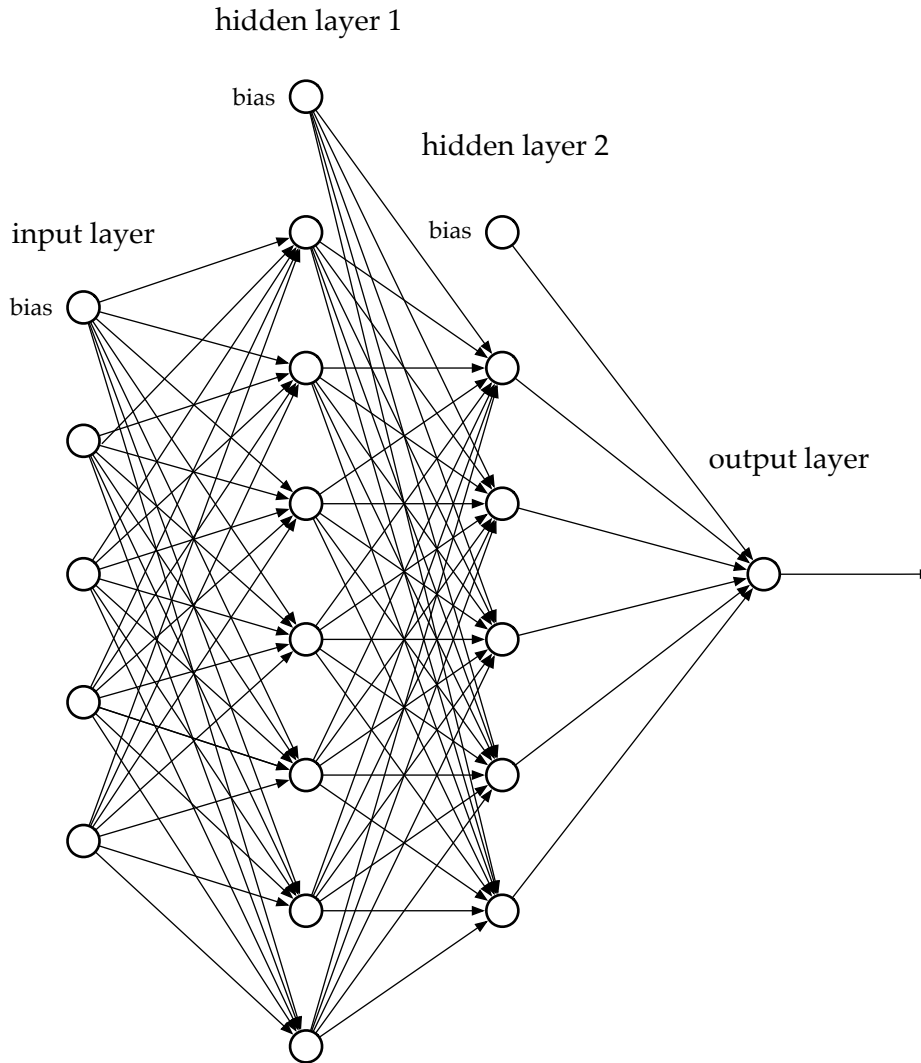
<sup>3</sup>If all neurons of an MLP are connected to each other, the network is said to be *fully-connected*.

of the whole network. If there is only one neuron in the output layer, the resulting MLP is analogous to nonlinear regression.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3)$$

$$\tanh(z) = 2\sigma(2z) - 1 \quad (4)$$

$$r(z) = \max(0, z) \quad (5)$$



**Figure 3:** A diagram of the architecture of a multilayer perceptron. 4 inputs, 2 hidden layers with 7 and 5 neurons respectively, and an output layer of one neuron.

I have now described the basic architecture of an MLP but for the reader it is yet unclear how the network actually learns a representation (i.e. a function) from its input data. It turns out that MLPs are really suitable for approximating functions: the results of Hornik et. al. (1989) and Hornik (1991) show that MLPs are universal approximators, which means that theoretically they can approximate virtually any measurable function to an arbitrary level of accuracy. However, to be able to learn an

approximation, the neural network must somehow be trained. This is done with an algorithm called *backpropagation*.

## 2.3 The Backpropagation Algorithm

The backpropagation training algorithm was introduced by Rumelhart et. al. (1986). The idea of the algorithm is to make the training of the MLP an iterative process where the parameters of the network are updated during the iterations. At each iteration, the inputs are passed through the network and the output value is calculated with the current parameter values of the network. This phase of the process is called the *forward pass*. After this, the output error (i.e. the error between the predicted value and the true desired value) is calculated, and the network parameters are updated to make the error smaller. The next sections will provide more details about the forward pass and parameter optimization.

### 2.3.1 The Forward Pass

The precise formulation for forward pass can be written as follows. In the input layer, we have the input vector  $\mathbf{x} \in \mathbb{R}^d$  and the bias term  $b^{(1)}$  (the bias terms are usually initialized to be 0 at the beginning of the training). After this, we have the input weights in the weight matrix  $\mathbf{W}^{(1)} \in \mathbb{R}^{m_1 \times d}$  where  $m_1$  is the size of the first hidden layer  $\mathbf{h}^{(1)}$ .<sup>4</sup> Then, the input elements are passed to each neuron in the first hidden layer as weighted sums. Thus, the value of each neuron in the first hidden layer can be written as

$$\begin{aligned} h_1^{(1)} &= \omega(b^{(1)} + W_{1,1}^{(1)}x_1 + W_{1,2}^{(1)}x_2 + \dots + W_{1,d}^{(1)}x_d) \\ h_2^{(1)} &= \omega(b^{(1)} + W_{2,1}^{(1)}x_1 + W_{2,2}^{(1)}x_2 + \dots + W_{2,d}^{(1)}x_d) \\ &\vdots \\ h_{m_1}^{(1)} &= \omega(b^{(1)} + W_{m_1,1}^{(1)}x_1 + W_{m_1,2}^{(1)}x_2 + \dots + W_{m_1,d}^{(1)}x_d) \end{aligned}$$

where  $\omega$  is the activation function. Note that it is possible for different layers to have different types of activation functions. Now, after the first hidden layer the process continues similarly as before, with the difference that now the inputs to the second hidden layer are the values (or outputs) of the neurons in the first hidden layer:

$$\begin{aligned} h_1^{(2)} &= \omega(b^{(2)} + W_{1,1}^{(2)}h_1^{(1)} + W_{1,2}^{(2)}h_2^{(1)} + \dots + W_{1,m_1}^{(2)}h_{m_1}^{(1)}) \\ h_2^{(2)} &= \omega(b^{(2)} + W_{2,1}^{(2)}h_1^{(1)} + W_{2,2}^{(2)}h_2^{(1)} + \dots + W_{2,m_1}^{(2)}h_{m_1}^{(1)}) \\ &\vdots \\ h_{m_2}^{(2)} &= \omega(b^{(2)} + W_{m_2,1}^{(2)}h_1^{(1)} + W_{m_2,2}^{(2)}h_2^{(1)} + \dots + W_{m_2,m_1}^{(2)}h_{m_1}^{(1)}) \end{aligned}$$

where  $\mathbf{W}^{(2)} \in \mathbb{R}^{m_2 \times m_1}$  is the weight matrix for the first hidden layer and  $m_2$  is the size of the second hidden layer  $\mathbf{h}^{(2)}$ . The process continues in a similar way until the output layer is reached. The whole

---

<sup>4</sup>The initial weights are usually drawn from standard normal distribution.



forward pass can be conveniently expressed in matrix notation as

$$\begin{aligned} \mathbf{h}^{(1)} &= \omega(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}) \\ \mathbf{h}^{(2)} &= \omega(\mathbf{b}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}^{(1)}) \\ &\vdots \\ \mathbf{h}^{(n)} &= \omega(\mathbf{b}^{(n)} + \mathbf{W}^{(n)}\mathbf{h}^{(n-1)}) \\ \hat{y} &= \Theta(b^{(out)} + \mathbf{w}^T \mathbf{h}^{(n)}) \end{aligned}$$

where  $n$  is the total number of hidden layers,  $b^{out}$  is the bias term for the output,  $\mathbf{w} \in \mathbb{R}^{m_n}$  is the vector of output weights,  $\Theta$  is the activation function of the output layer and  $\hat{y}$  is the output of the network. The network output can then be written as a function

$$\hat{y} = f(\mathbf{x}; \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(n)}, \mathbf{w}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(n)}, b^{(out)}) \quad (6)$$

### 2.3.2 Parameter Optimization with Stochastic Gradient Descent

As mentioned earlier, after completing the forward pass the output error is calculated with some cost function. There are many kinds of different cost functions that could be used but the most usual choice for regression-like tasks is the squared error

$$L(\mathbf{x}, y; \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2 = (y - \hat{y})^2 \quad (7)$$

where  $\boldsymbol{\theta}$  denotes the parameters that are used to calculate the prediction  $\hat{y}$ , and  $y$  is the desired true value. The total cost of the training set can now be defined as

$$\mathcal{E}(f(\cdot)|\mathbb{X}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(\mathbf{x}^{(i)}))^2 = \frac{1}{N} \sum_{i=1}^N L(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) = g(\boldsymbol{\theta}) \quad (8)$$

where  $\mathbb{X}$  denotes the data of a training set of  $N$  instances,  $y^{(i)}$  is the true value of instance  $i$  and  $f(\mathbf{x}^{(i)})$  is the prediction from the data of instance  $i$ .<sup>5</sup>

The objective is to minimize the cost function, and this can be done by calculating the gradient of the cost function with respect to all of the parameters of the network; the gradient indicates how much each parameter contributed to the error. After this, the parameters are updated as shown in (9). This is called the *gradient descent* algorithm.

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \nabla g(\boldsymbol{\theta}^{(k)}) \quad (9)$$

Here  $\alpha$  is a hyperparameter<sup>6</sup> called the *learning rate* and  $k$  denotes the number of the current forward pass. *Stochastic* gradient descent means that at each iteration, the gradient descent algorithm uses the training data in smaller subsets called *mini-batches*, which are drawn from the training set that has been randomly shuffled. This usually speeds up the convergence of the cost function and thus reduces the training time.

<sup>5</sup>One instance means the data of one observation.

<sup>6</sup>An external parameter that is set by the user, not a parameter for the network to learn.

One more crucial aspect related to the training algorithm is to know when to stop iterating. There are several stopping criteria that can be used to indicate when to stop the training. The most usual ones are

1. Set up a maximum number of iterations.
2.  $\Delta g(\boldsymbol{\theta}) = g(\boldsymbol{\theta}^{(k+1)}) - g(\boldsymbol{\theta}^{(k)}) < \delta$  for some  $\delta > 0$ , i.e. the iteration will stop when the reduction in the cost function becomes smaller than some threshold.

## 2.4 Neural Networks in Option Pricing

As mentioned in the introduction section, several studies have experimented with neural networks in option pricing. Among the neural network models used in these studies, the MLP has been by far the most popular one: all the aforementioned studies have used MLP architectures in their analysis. Some authors have also experimented with other neural network models. For example, Hutchinson et. al. (1994) compared three different neural network models: an MLP, a radial-basis function network and a projection pursuit network, and found that MLP clearly had the best performance. The high performance combined with the straightforward architecture of the MLP is probably the main reason why the later studies (see e.g. Anders et. al (1996), Gençay and Salih (2003), Yao et. al. (2000)) have focused mainly on MLPs.

However, many of the previous studies have used rather small, or *shallow*, network structures in their MLPs: Hutchinson et. al. (1994) had four neurons in one hidden layer and used two input variables; Bennell and Sutcliffe (2004) experimented with combinations of three to five neurons in one layer with three to seven input variables; Anders et. al. (1996) used an MLP with sparse connectivity (i.e. all inputs were not connected to all neurons in the hidden layer) and had three neurons in one hidden layer with four input variables. Despite of their small size, the performance of these networks has been rather high. Because deep models are generally more capable of learning higher level (i.e. more abstract) patterns and structures from data (see e.g. LeCun et. al. (2015)), it is reasonable to assume that a deeper (i.e. more hidden layers) and wider (i.e. more neurons in each layer) neural network will perform at least as well as the shallower networks in the previous studies. With today's computational power it is relatively fast to train significantly bigger networks: for example, an MLP with hundreds of neurons and tens of layers can be trained with a basic laptop in a reasonable amount of time.

## 3 The Data

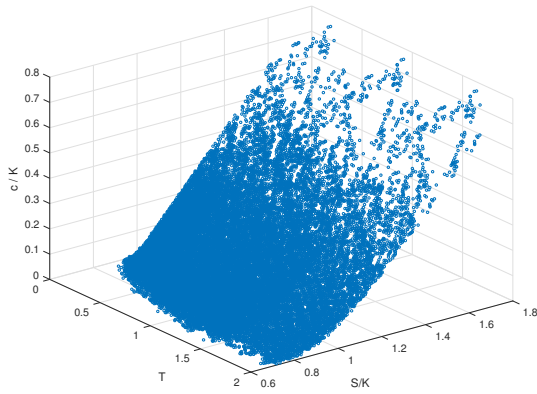
The full dataset consists of daily DAX 30 call option closing prices ranging from January 1st 2013 to September 19th 2017 and the corresponding daily DAX 30 spot prices. All data was collected from Datastream. In total there are 1231 days with observations. The dataset contains a total of 668 different call options with 25 different strike prices ranging from 3500 to 15500. The total number of observations for options prices is 130132.

I filtered the data according to the suggested exclusion criteria of Anders et. al. (1996) in order to remove non-representative observations. I filtered out observations where

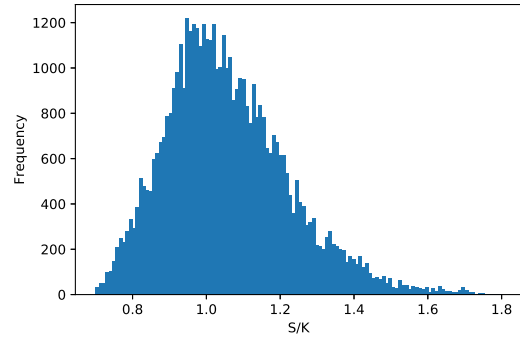
1. The call option is traded below 10 index points.

2. Time to maturity is more than two years.
3. At initialization, the option is either deep out-of-the-money (OTM) or deep in-the-money (ITM):  $\frac{S}{K} < 0.70$  or  $\frac{S}{K} > 1.20$ . For this rule, the whole time series for the option in question was removed. ( $S$  denotes the spot,  $K$  denotes the strike price of the option.  $\frac{S}{K}$  is the *moneyness* of the option.)
4. The lower bound condition for call option price ( $c \geq S - Ke^{-rT}$ ) is violated. ( $c$  is the call price,  $r$  is the risk-free rate and  $T$  time to maturity.)

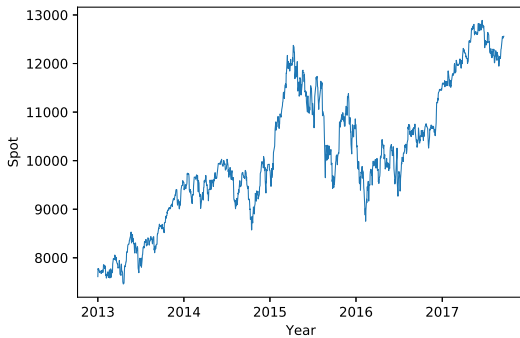
After filtering, the dataset consists of 48486 observations. Figure 4a shows the call prices of the final dataset normalized by the strike prices plotted in a maturity-moneyness space: it can be seen that the final dataset represents well both the ITM and OTM side for all maturities. Figure 4b shows the distribution of the moneyness values in the final dataset. Note the slight skew to the ITM side (average moneyness is 1.057). The average time to maturity of the options is 0.770 years (in this paper I use 252 as the number of trading days per year). Figure 4c shows the spot movement during the time period.



(a) Final dataset: scatter plot of call prices normalized by strike in moneyness-maturity space.



(b) Final dataset: histogram of the moneyness values.



(c) DAX 30 daily spot price between 1/1/2013 and 19/9/2017.

**Figure 4:** Graphs of the data.

The final dataset was further divided to two sets: the training set and the test set. The time period of the training set is 28/1/2013 to 29/1/2016 (a total of 785 days with observations), and the training

set includes 269 different options with 18 different strikes and a total of 32661 observations. The yearly volatility of the spot was 18.88% during the train period. For the test set, the time period is 1/2/2016 to 14/9/2017 (a total of 424 days with observations), and it includes 156 different options with 15825 observations and 14 different strikes. During the test period, the spot volatility was 18.11%. The time periods for the training set and test set were chosen so that the size of the test set is sufficiently large. The neural network will be trained with the train set and then its out-of-sample performance will be evaluated with the test set. The performance measures will be introduced in the next chapter.

## 4 Model Specification

### 4.1 Neural Network Structure

The model that I will use in the analysis is an MLP with four hidden layers and implemented with *Theano*<sup>7</sup>, a Python library that is widely used in deep learning applications. The sizes of the hidden layers are 100, 80, 60, and 40 respectively, and all neurons in the hidden layers use the logistic sigmoid (3) as activation function. The sizes of the hidden layers and the activation function were chosen based on experiments with the performance of the network on the training set. The output layer consists of one neuron which has the softplus function (11) as activation function. I chose softplus because it maps the whole real line to nonnegative real numbers (option prices cannot be negative). The parameter optimization algorithm is stochastic gradient descent with a mini-batch size of 128. The number of training epochs (i.e. iterations) is 1000. There are two inputs: the moneyness ( $S/K$ ) and the time to maturity ( $T-t$ ), which are the same inputs as used by Hutchinson et. al. (1994). The output of the network corresponds to the call price normalized by the strike price. This input-output structure is based on the assumption that the functional shape  $f$  of the neural network pricing formula is homogenous of degree one in  $S$  and  $K$ , and Garcia and Gençay (2000) showed that the use of this homogeneity assumption (10) is crucial for the superior performance of the neural network model.

$$c = f(S, K, T-t) \Rightarrow \frac{c}{K} = f(S/K, 1, T-t) \quad (10)$$

$$s(z) = \ln(1 + e^z) \quad (11)$$

### 4.2 Performance Measures

#### 4.2.1 Pricing Performance Measures

The performance of the MLP will be compared to the basic Black-Scholes model. The Black-Scholes call price at time  $t$ , denoted by  $c_{BS}$  is

$$c_{BS}(t) = S_t \phi(d_1) - K e^{-r(T-t)} \phi(d_2) \quad (12)$$

where

$$d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})(T-t)}{\sigma \sqrt{T-t}} \quad (13)$$

$$d_2 = d_1 - \sigma \sqrt{T-t} \quad (14)$$

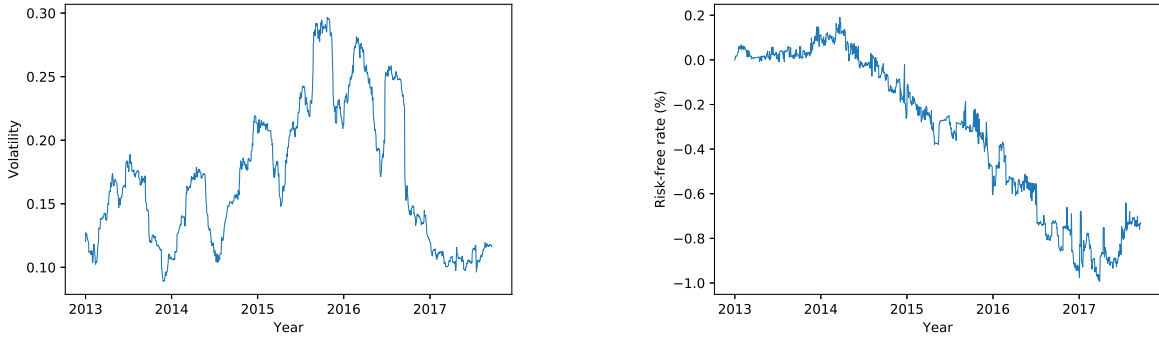
---

<sup>7</sup>See <http://deeplearning.net/software/theano/> for further information about Theano.

and  $\phi(\cdot)$  is the cumulative distribution function of standard normal distribution,  $S(t)$  is the spot price at time  $t$ ,  $K$  is the strike price,  $T$  is the time to maturity of the option,  $\sigma$  is the volatility and  $r$  is the risk-free rate. Note that there are no straight values to be assigned for the parameters  $\sigma$  and  $r$  so they have to be estimated. I estimate these parameters similarly to Hutchinson et. al. (1994):  $r$  is estimated with the 3-month German government bond yield, and the volatility will be estimated with the rolling standard deviation with a window size of 60

$$\hat{\sigma} = \frac{s}{\sqrt{60}} \quad (15)$$

where  $s$  is the standard deviation of the continuously compounded daily returns of the spot price of DAX 30 from the last 60 days.



(a) Estimated volatility with a window size of 60.

(b) Risk-free rate (3-month German government yield).

**Figure 5:** Estimated volatility and risk-free rate between 1/1/2013 and 19/9/2017

Let  $c_{NN}$  denote the call price given by the neural network and  $c$  denote the real market call price. The pricing performance will be evaluated with the following error functions, also used by Bennell and Sutcliffe (2004): mean error (16), mean absolute error (17), mean squared error (18) and root mean squared error (19). From these measures, (16) and (19) are probably the most intuitive: (16) can be thought as a measure of average over/underpricing and (19) as the pricing error as a percentage from the strike price (this is based on the homogeneity assumption). (17) and (18) can be used to sanity-check the results.

$$\text{ME} = \frac{1}{N} \sum_{i=1}^N (c_{NN}^{(i)} - c^{(i)}) \quad (16)$$

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |c_{NN}^{(i)} - c^{(i)}| \quad (17)$$

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (c_{NN}^{(i)} - c^{(i)})^2 \quad (18)$$

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (c_{NN}^{(i)} - c^{(i)})^2} \quad (19)$$

Here  $N$  denotes the size of the test set.

### 4.2.2 Delta Hedging Performance Measure

In this part I will follow the methodology of Hutchinson et. al. (1994) and measure the delta-hedging performance of the neural network with a *tracking error*. The idea of delta-hedging is to set up a hedging portfolio that offsets the risk from an option position, and this hedging portfolio works correctly if the combined value of the option position and the hedging portfolio is zero at the expiration of the option. The Black-Scholes model assumes continuous delta-hedging, which is not possible in real life and this makes the combined value of the option position and the hedging portfolio different from zero at expiration. This is called the delta-hedging tracking error. Due to the discrete delta-hedging, there will always be some tracking error and the delta-hedging performance of the neural network will be evaluated by comparing if the neural network tracking error is lower than with Black-Scholes. The delta-hedging strategy for the neural network model is as follows:

Denote the total value of the delta hedging positions at time  $t$  as  $V(t)$ .

$$V(t) = V_S(t) + V_B(t) + V_c(t) \quad (20)$$

Here  $V_S(t)$  is the value of the spot position,  $V_B$  is the value of the bond position which is used to finance the spot position and  $V_c$  is the value of the call position. At time  $t = 0$  we take the positions

$$V_c(0) = -c_{BS}(0) \quad (21)$$

$$V_S(0) = \Delta_{NN}(0)S(0) \quad (22)$$

$$V_B(0) = -(V_S(0) + V_c(0)) \quad (23)$$

$$\Delta_{NN}(0) = \frac{\partial c_{NN}(0)}{\partial S} \quad (24)$$

$$\Rightarrow V(0) = 0 \quad (25)$$

In other words, the strategy writes one call option (21), goes long for the spot (22) and goes short for a bond (23) to get the required amount of cash for the spot position. Note that  $\Delta_{NN}$  is the partial derivative of the trained network pricing formula with respect to the spot. The calculation of the partial derivative leads to long chain rules, and this can indeed be calculated analytically with the computational graph of Theano.

At all times  $t$  between the initialization of  $V$  and expiration, the spot and bond positions are updated daily as follows:

$$V_S(t) = \Delta_{NN}S(t) \quad (26)$$

$$\Delta_{NN}(t) = \frac{\partial c_{NN}(t)}{\partial S} \quad (27)$$

$$V_B(t) = e^{r\tau}V_B(t - \tau) - S(t)(\Delta_{NN}(t) - \Delta_{NN}(t - \tau)) \quad (28)$$

where  $\tau$  is the time step which is one day in this case. The tracking-error of the portfolio is then

$$\epsilon = V(T) \quad (29)$$

For delta-hedging with Black-Scholes,  $\Delta_{NN}$  is replaced with

$$\Delta_{BS} = \frac{\partial c_{BS}}{\partial S} = N(d_1) \quad (30)$$

## 5 Empirical Results and Analysis

### 5.1 Pricing Performance Results

**Table 1:** The following tables show the out-of-sample pricing performance measures for the neural network model (NN) and Black-Scholes (BS) for the test set. The measures are calculated with the call prices normalized by strike. In the tables the values of ME and MAE are multiplied by  $10^3$ , MSE by  $10^4$  and RMSE by  $10^2$  to improve readability. ITM indicates that  $S/K > 1.02$ , OTM that  $S/K < 0.98$  and NTM (near-the-money) that  $0.98 \leq S/K \leq 1.02$ . Similarly, short term indicates that  $T \leq 1/12$ , medium term that  $1/12 < T \leq 1/2$  and long term that  $T > 1/2$ . From the error functions, RMSE is probably the most intuitive as it tells the pricing error as a percentage of the strike price. The test set contains 15825 daily observations of call prices and the corresponding spot prices between 1/2/2016 and 14/9/2017.

a) Whole test set						
Model	Observations	ME	MAE	MSE	RMSE	
BS	15538	2.011	10.914	2.132	1.460	
NN		-0.665	5.348	0.510	0.714	

b) Whole test set partitioned by moneyness						
Model	Moneyiness	Observations	ME	MAE	MSE	RMSE
BS	ITM	8896	-4.065	9.295	1.473	1.214
NN			-1.837	5.036	0.468	0.684
BS	NTM	908	2.552	11.376	2.160	1.470
NN			0.848	7.802	0.909	0.953
BS	OTM	5734	11.351	13.354	3.150	1.775
NN			0.913	5.444	0.512	0.716

c) Whole test set partitioned by maturity						
Model	Maturity	Observations	ME	MAE	MSE	RMSE
BS	SHORT	1131	-0.786	1.975	0.071	0.267
NN			0.508	7.189	0.716	0.846
BS	MEDIUM	6718	-1.022	6.285	0.645	0.803
NN			1.911	4.979	0.405	0.636
BS	LONG	7689	5.072	16.273	3.735	1.933
NN			-3.088	5.400	0.571	0.756

d) Short term options						
Model	Moneyiness	Observations	ME	MAE	MSE	RMSE
BS	ITM	877	-1.302	1.775	0.051	0.225
NN			-2.748	5.864	0.488	0.698
BS	NTM	108	-0.562	2.667	0.126	0.356
NN			11.846	11.869	1.551	1.245
BS	OTM	146	2.146	2.665	0.154	0.392
NN			11.683	11.683	1.473	1.214

e) Medium term options						
Model	Moneyiness	Observations	ME	MAE	MSE	RMSE
BS	ITM	4385	-3.935	5.988	0.526	0.725
NN			0.331	4.112	0.298	0.546
BS	NTM	441	0.880	7.530	0.903	0.950
NN			3.340	6.876	0.682	0.826
BS	OTM	1892	5.286	6.685	0.862	0.928
NN			5.238	6.544	0.589	0.767

f) Long term options						
Model	Moneyiness	Observations	ME	MAE	MSE	RMSE
BS	ITM	3634	-4.889	15.099	2.960	1.720
NN			-4.233	5.951	0.668	0.817
BS	NTM	359	5.542	18.720	4.317	2.078
NN			-5.523	7.716	0.995	0.997
BS	OTM	3696	14.820	17.190	4.440	2.107
NN			-1.726	4.634	0.435	0.659

Table 1a shows that the neural network is superior by all measures, and this result is consistent with the findings of Hutchinson et. al. (1994) and Anders et. al. (1996). In both studies, the authors analyzed the pricing performance with the whole test set similarly as in Table 1a, without separating the measures by moneyness or maturity. The values of ME indicate that the neural network is on average slightly underpricing whereas Black-Scholes is overpricing. Similar findings regarding the underpricing behavior of the neural network were also acquired by Malliaris and Salchenberger (1993). The underpricing bias of the neural network can be seen in Figure 6a. However, it is of interest to analyze the pricing performance in a more detailed manner by calculating the performance measures based on different moneyness levels and maturities separately.

When the whole test set is partitioned by moneyness (Table 1b), it can be seen that the neural network model is superior at all moneyness levels, especially at OTM. This is consistent with the results of previous studies: the results of Malliaris and Salchenberger (1993), Gençay and Salih (2003) and Bennell and Sutcliffe (2004) also indicate that the neural network model is superior for OTM options. The reason for the inferiority of Black-Scholes is the heavy overpricing of OTM options. This kind of behavior of the Black-Scholes model has been found also in previous studies (see e.g. Gençay and Salih (2003)). The clear overpricing of OTM options of Black-Scholes can be seen in Figure 6b. In addition, based on the mean errors both of the models seem to underprice ITM and overprice NTM (near-the-money) options and the degree of mispricing of Black-Scholes is clearly higher at both moneyness levels.

When the partitioning is done by maturity (Table 1c), the results differ from the previous ones. The neural network model is still superior for medium-term and long-term options but for short-term options Black-Scholes seems to be superior. Figure 7a and Figure 7b illustrate this result quite clearly: it can be seen that Black-Scholes is much more stable than the neural network model for short-term maturities across all moneyness levels.

More detailed results for short-term options can be seen in Table 1d. The values indicate that for short-term maturities Black-Scholes is indeed superior by all measures at all moneyness levels. For NTM options this is a bit surprising as in previous studies the neural network model has been found to perform better than Black-Scholes for short-term near-the-money options (see e.g. Bennell and Sutcliffe (2004)). The performance of the neural network is inferior because it clearly overprices short-term NTM and OTM options.

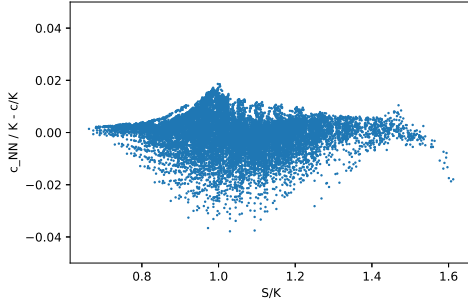
For medium-term maturities (Table 1e), the neural network seems to perform better but the differences are quite small. For medium-term NTM options, the neural network overprices more than Black-Scholes, and generally it seems that the degree of overpricing for the neural network increases when moving from ITM towards OTM. For long term maturities (Table 1f) the performance of the neural network is clearly superior at all moneyness levels even though it seems to underprice especially long-term ITM and NTM options.

To summarize the pricing performance results, it seems that Black-Scholes is superior with shorter maturities and the neural network with longer maturities. A possible explanation for this could be the use of the 3-month German bond yield as the risk-free rate for Black-Scholes but it cannot explain the good short-term performance of Black-Scholes completely as the short term maturities are less than one month. Regarding different moneyness levels, the values of RMSE indicate that the neural network model generally has better performance for ITM and OTM options compared to NTM options and the clearest superiority for OTM options. Figure 7a shows quite clearly how the absolute error of Black-Scholes increases with respect to maturity, and also it can be seen that Black-Scholes has

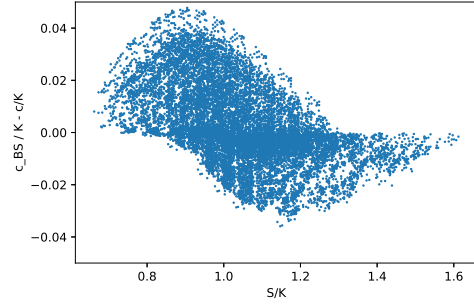


problems in pricing long-term near-the-money options. In turn, Figure 7b shows that the neural network model does not perform so well with short maturities but the performance is quite stable with medium and long-term options. However, the surface shows that also the performance of neural network becomes unstable with very long maturities.

There are a couple of aspects about these results that should be pointed out. First, the performance of the neural network could most probably be improved by using more input variables and more advanced optimization algorithms such as AdaGrad (see Duchi et. al. (2011)), or Adam (see Kingma and Ba (2014)). Also different kinds of regularization techniques such as dropout (see Srivastava et. al. (2014)) that prevent the network from overfitting could improve the performance. Secondly, the Black-Scholes parameters were not optimized in this analysis. The risk-free rate parameter could be optimized for example by interpolating from the yield curve. The volatility parameter could be optimized simply by experimenting with different sizes of the time interval. Moreover, the dividend yield of DAX 30 was not accounted for in this analysis for simplicity, and including the dividend yield would most probably improve the performance of the Black-Scholes model.

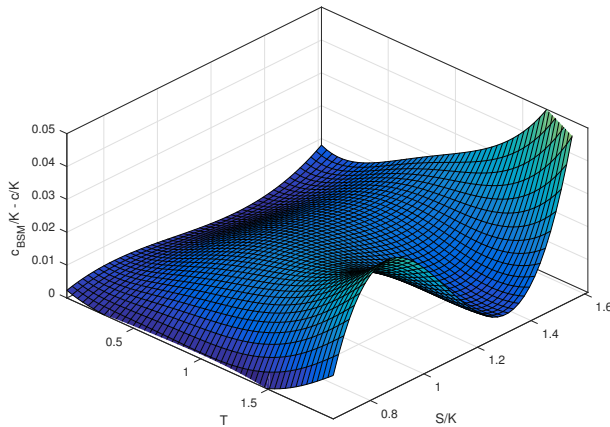


(a) Neural network pricing error:  $\frac{c_{NN}}{K} - \frac{c}{K}$

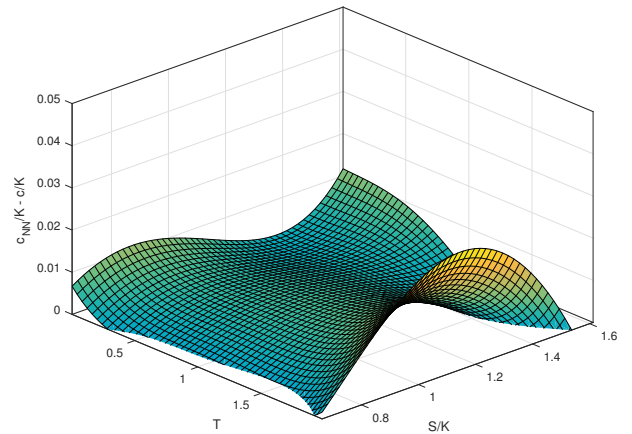


(b) Black-Scholes pricing error:  $\frac{c_{BS}}{K} - \frac{c}{K}$

**Figure 6:** Pricing error scatter plots.



(a) Black-Scholes absolute pricing error:  $|\frac{c_{BS}}{K} - \frac{c}{K}|$



(b) Neural network absolute pricing error:  $|\frac{c_{NN}}{K} - \frac{c}{K}|$

**Figure 7:** Absolute pricing error surfaces.

## 5.2 Delta Hedging Performance Results

In total there were 134 options for which hedging was performed. I will evaluate the performance with a two-sided paired t-test for the absolute values of the tracking errors. The setup is the following:

$$D_i = |\epsilon_{NN}^{(i)}| - |\epsilon_{BS}^{(i)}|$$

$$\bar{D} = \frac{1}{N} \sum_{i=1}^N D_i$$

$$H_0 : \mu_D = 0$$

$$H_1 : \mu_D \neq 0$$

where  $\mu_D$  is the true mean of the difference  $D$ . Note that the results of this t-test should be evaluated with caution as the observations are not statistically independent because of the overlapping spot paths.

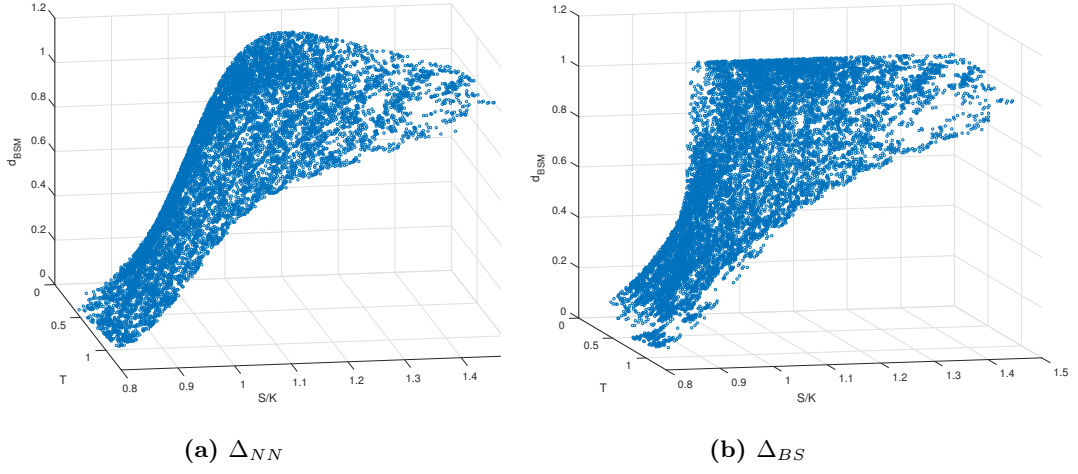
**Table 2:** Delta hedging performance: paired t-tests for the absolute tracking errors for the whole test set. The absolute tracking error of the neural network is on the left hand side in the difference of the pairs. Tracking error means the total value of the delta hedging positions at the expiration of the option. ITM means that  $S/K > 1$  and OTM that  $S/K < 1$  at the beginning of the hedging. The rightmost column shows the fraction of the sample in which the absolute tracking error of the neural network was smaller than the absolute tracking error of Black-Scholes.

Sample	t-statistic	p-value	Observations	% NN < BS
ITM	4.087	<0.001	70	28.57
OTM	-2.307	0.024	64	56.25
All	-0.120	0.904	134	41.79

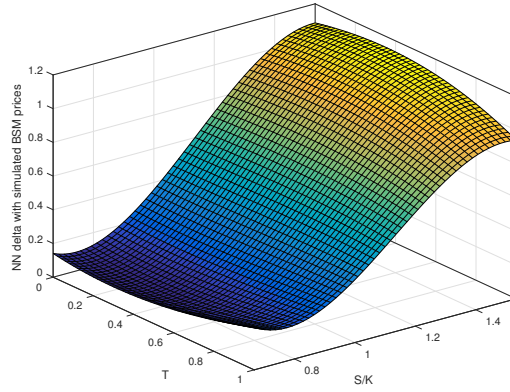
The values in Table 2 indicate that for the whole test set, neither of the models is superior. However, the hypothesis about the superior delta hedging performance of the neural network model is still partially valid: the t-statistic for OTM options implies the neural network is superior in delta hedging OTM options, and this result is significant at 5% level. For ITM options, Black-Scholes is superior at 0.1% level.

The reason for why the delta hedging performance of Black-Scholes is superior for ITM options shows clearly in Figure 8a. Firstly, some of the network deltas are greater than one at the ITM region (the delta of a call option should always be between 0 and 1, and Figure 8b illustrates this quite clearly). In addition, when moneyness increases, the network delta starts to decrease at the ITM region. This is not consistent with the theory as the delta of a call should be a monotonous function of the spot. Clearly, the neural network has not learned the true theoretical properties of the delta. This is a bit surprising result because as pointed out earlier, the deep neural network model should in theory be able to learn higher level properties from the data, and the delta is just this kind of property that is not explicitly present in the data but it is still implicitly "built in" to the option pricing mechanism.

In order to get a hint of whether this behavior is caused by the data or by the model, I simulated Black-Scholes prices and trained the network with them. Figure 9 shows the resulting neural network delta surface. There are still delta values above 1 but the requirement for monotonicity is much better



**Figure 8:** Black-Scholes delta and neural network delta.



**Figure 9:**  $\Delta_{NN}$  from simulated Black-Scholes prices.

satisfied. Based on this, it seems that the neural network would probably be able to learn the delta quite well from real data but the nature of the used dataset is such that the network is not able to learn the delta in such way that it would be fully consistent with the theory.

Because of the controversial delta hedging performance, my results are not truly consistent with previous literature: for example the delta hedging performance results of Hutchinson et. al. (1994) indicate that the neural network is clearly superior with a t-statistic of 3.78. However, Anders et. al. (1996) also found that their real data most probably contained such features that the neural network deltas were not fully consistent with theory but with simulated data the network produced deltas that were more consistent with the theory. Moreover, because of these results and also because the delta hedging performance has not been covered in many of the earlier studies, I would recommend further research that would focus solely on the delta hedging performance.

## 6 Conclusions

In this paper I analyzed whether a deep neural network model has better performances in pricing and delta hedging European-style call options than the Black-Scholes model. I trained a multilayer perceptron and tested the out-of-sample pricing and delta hedging performances with daily prices of call options written on DAX 30 between years 2013 and 2017. Pricing performance was measured with several error functions such as mean error and root mean squared error and delta hedging performance was measured as the total value of the dynamic delta hedging portfolio at the expiration of the option. The main hypotheses were that the neural network model outperforms Black-Scholes 1) in pricing performance and 2) in delta hedging performance.

Based on the results, I find that the first hypothesis regarding the pricing performance generally holds. However, more detailed analysis about the pricing performance in different moneyness levels and maturity times reveals that Black-Scholes outperforms the neural network model in pricing short-maturity options at all moneyness levels. For medium and especially long-term options, the pricing performance of the neural network is superior especially for out-of-the money options, and this result is consistent with previous literature.

The hypothesis about the delta-hedging performance holds partially as the Black-Scholes is superior in delta-hedging in-the-money options and the neural network is superior with out-of-the-money options. The main cause for this is that for in-the-money options, the neural network deltas are not fully consistent with the theory of options pricing. Because of these controversial results and the fact that delta hedging performance has not been covered as widely as pricing performance in the previous studies, I suggest further research that should focus solely on the delta hedging performance.

The neural network model which I used in this paper was considerably larger (i.e. wider and deeper) than the ones that have been used by the authors in previous literature. My results are generally in line with the previous studies but the deeper model that I used doesn't seem to have significantly higher performance compared to the shallow ones of the previous studies.

Furthermore, the performance of the neural network could probably be improved by using more input variables and more advanced optimization algorithms. Also different kinds of regularization techniques that prevent the network from overfitting could improve the performance. Moreover, the performance of the Black-Scholes model could be improved by optimizing the risk-free and volatility parameters and by including dividends in the model. Thus, regarding further research, in addition to focusing on the delta hedging performance I would suggest to compare a neural network to Black-Scholes in such way that both of the models are highly optimized. Also, neural networks should be compared to other parametric pricing models such as the Heston model (see Heston (1993)) and with different types of options, such as currency options. It would also be of interest to study how a neural network model performs when compared to numerical pricing methods for options that do not have closed-form solutions.

## 7 References

- [1] Amilon, H. 2003. A Neural Network Versus Black-Scholes: A Comparison of Pricing and Hedging Performances. *Journal of Forecasting* 22, 317-335.
- [2] Anders, U., Korn, O., Schmitt, C. 1996. Improving the pricing of options: a neural network approach. *ZEW Discussion Papers* 96-04.
- [3] Bennell, J., Sutcliffe, C. 2004. Black-Scholes versus Artificial Neural Networks in Pricing FTSE 100 Options. *Intelligent Systems in Accounting, Finance and Management* 12, 243-260.
- [4] Black, F., Scholes, M. 1973. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy* 81, 637-654.
- [5] Duchi, J., Hazan, E., Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, 2121-2159
- [6] Fukushima, K. 1980. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics* 36, 193-202.
- [7] Garcia, R., Gençay, R. 2000. Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics* 94, 931-15.
- [8] Gençay, R., Salih, A. 2003. Degree of Mispricing with the Black-Scholes Model and Nonparametric Cures. *Annals of Economics and Finance* 4, 73-101.
- [9] Hanke, M. 1999. Neural Networks vs. Black/Scholes: An Empirical Comparison of Two Fundamentally Different Option Pricing Methods. *Journal of Computational Intelligence in Finance* 7, 26-34.
- [10] Heston, S. L. 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies* 6(2), 327-343.
- [11] Hornik, K. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks* 4, 251-257.
- [12] Hornik, K., Stinchcombe, M., White, H. 1989. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* 2, 359-366.
- [13] Hutchinson, J., Lo, A., Poggio, T. 1994. A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning. *The Journal of Finance* 49, 851-889.
- [14] Kingma, D., Ba, J. 2014. Adam: A method for stochastic optimization. [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
- [15] LeCun, Y., Bengio, Y., Hinton, G. 2015. Deep learning. *Nature* 521, 436-444.
- [16] Malliaris M, Salchenberger L. 1993. A neural network model for estimating option prices. *Journal of Applied Intelligence* 3, 193-206.
- [17] McCulloch, W., Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5, 115-133.

- [18] Merton, R. 1973. Theory of Rational Option Pricing. *The Bell Journal of Economics and Management Science* 4, 141-183.
- [19] Rosenblatt, F. 1958. A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65, 386-408.
- [20] Rumelhart, D., Hinton, G., Williams, R. 1986. Learning representations by back-propagating errors. *Nature* 323, 533-536.
- [21] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1929-1958.
- [22] Yao J, Li Y, Tan L. 2000. Option price forecasting using neural networks. *Omega* 28, 455-466.